ISSN 0265-2919

80p

THE HOME COMPUTER COURSE 17

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Hardware

Outer Limits Even an inexpensive home computer like the Sinclair ZX81 can evolve into a sophisticated data processing system 326

Tandy MC-10 This compact machine offers good colour at a low price 330

Software

Top Gear We explore a variety of techniques for writing more efficient Basic and speeding up your programming 328

Basic Programming

Changing Places We look at ways to manipulate the database we have created in our Basic Programming course 336

Insights

Sitting Pretty Alternative designs for computer keyboards and screens could make them more pleasant to use 321

Hot Rods Not all joysticks are attached to a fixed base unit 332

Passwords To Computing

Track Record A Disk Operating System keeps a record of the track and sector numbers of every block of data 324

Pioneers In Computing

Konrad Zuse Like other early computers, Zuse's machines were developed for military purposes 340

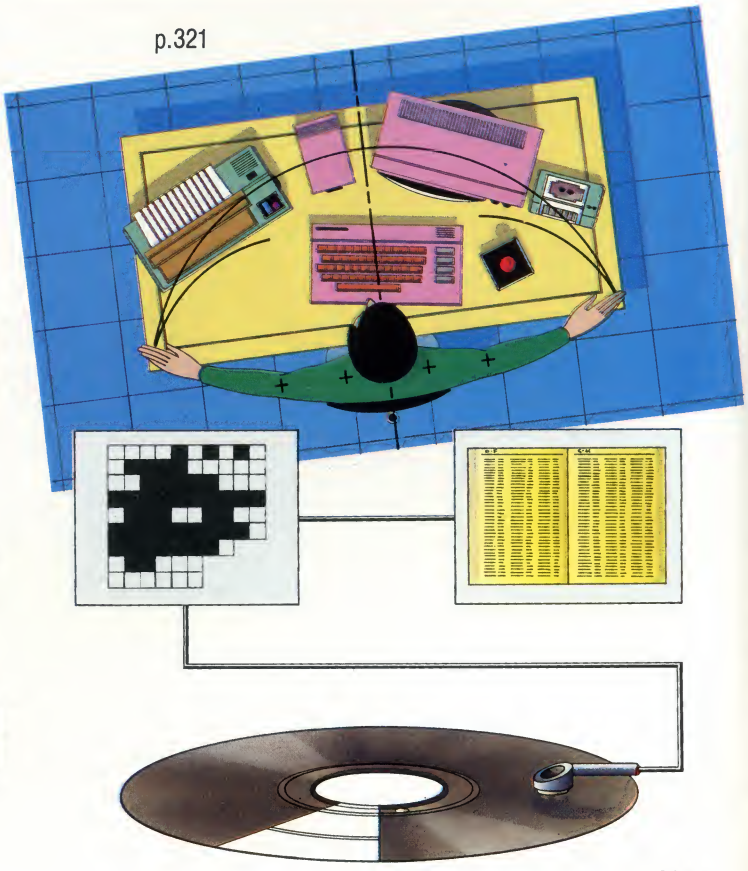
Sound And Light

Sound Systems... The Light Program An introduction to the graphics facilities on the BBC Micro and a further look at sound on the Vic-20 334

Next Week

- The Apple II has been described as the Volkswagen of microcomputers. We look at the latest model, and explain the reasons for its cult-like following
- BASIC is the most popular language for home computing, but by no means the only one, or indeed the best. We look at several alternatives
- Hard disks are faster than floppy disks, and have greater capacity. Soon they'll be available for home computers. We examine how they work

p.321



p.324

Editor Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistants** Liz Dixon, Safu Maria Gilbert; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Lisa Kelly, Steven Colwill, Richard King, Geoff Nairns; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Consultant** David Tebbutt; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© 1983 by Orbis Publishing Ltd:** **Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER COURSE – Copies are obtainable by placing a regular order at your newsagent.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595. Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

Sitting Pretty

'Ergonomics' is the science of making machines more pleasant to use. With computers, research has been concentrated on the screen and keyboard

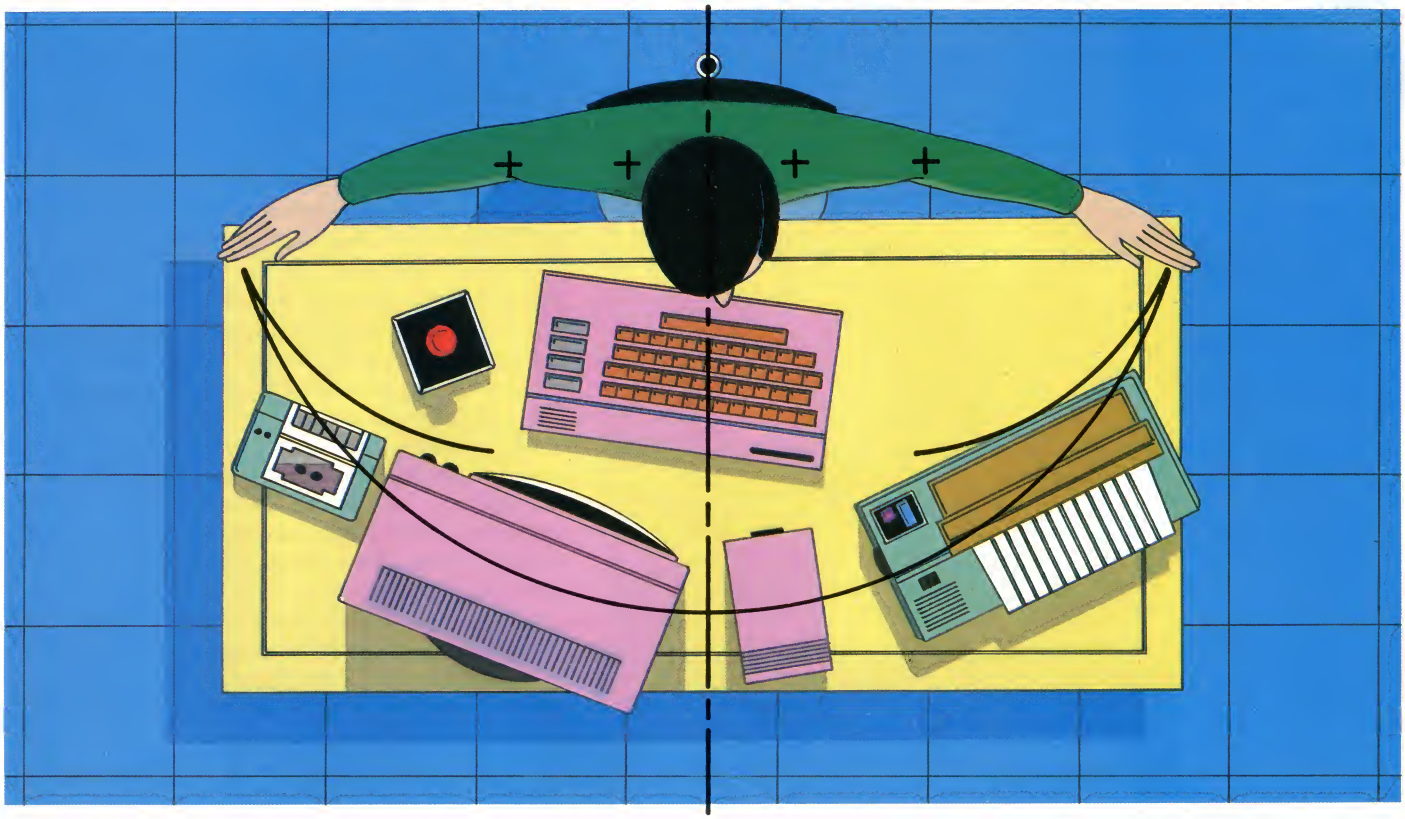
There are two aspects of design: aesthetics, or beauty in form and appearance; and ergonomics, which is the study of the relationship between workers and their environment. No matter how well something functions, we will be unhappy using it if it is ugly in appearance. Similarly, the environment in which we are working must not be distracting or uncomfortable.

As a factor in the choice of which microcomputer to buy, the ergonomic quality will probably be less of a consideration than the price and performance of the machine. It is, however, worthwhile to give some thought to the physical environment in which you use the computer. First of all, do you work at something that resembles an office workstation, with adequate desk space on a working surface set at the right height for you? Or do you simply plug your home computer into the family television set and work with it on your lap, or, worse still, lying on the ground in front of the television set?

Computer programming is quite complicated enough on its own, without making it more difficult by working in a completely unsuitable environment. There are many ways in which you can create a more comfortable workstation. Let us start by considering what can be done to make the screen more comfortable to read. If you are using a domestic television set, then you will be unable to benefit from recent developments that help reduce or eliminate screen glare in monitors. These include filters to minimise reflection and specially coloured screen phosphors. But you can improve the quality of display on a television set by placing a filter over the screen. Simple coloured filters are easy enough to obtain, and it is also possible to use a polarising filter, which eliminates reflections. These methods help to achieve high contrast at low brightness levels, and thus avoid unnecessary eyestrain.

External lighting levels are also important. When working at night it is considerably better to use a low-set desk lamp that illuminates the keyboard and any notes from which you are working, but leaves the screen in comparative darkness. The distance from eye to screen is also important — the body should be approximately an arm's-length from the screen. The display itself should be tiltable, so that the plane of the screen is at 90 degrees to the line from your eye to the

Body Language
While the human body may vary in size and shape, the proportions stay fairly constant, as any student of figure drawing soon realises. The study of ergonomics makes use of this consistency to define general rules for laying out working environments. With a home computer or a VDU these suggest that the screen should be at arm's length (to minimise changes in eye focus when looking backwards and forwards between screen and source document). The position of the keyboard is also dictated by these same rules



ROY INGRAM

centre of the screen. This can be easily achieved by placing a book or two under the front of the set. However, you may encounter another problem at this point — your own reflection in the screen. A filter with a matt surface will remove this quite effectively.

Once the screen has been made comfortable to use, we can go on to consider the physical layout and attributes of the keyboard. The most important factors are the height of the keys above the desk on which the keyboard is placed, and the angle of the rows of keys relative to each other. In ideal circumstances, the keyboard will be low enough for the operator's wrists and forearms to rest flat on the desk in front of it, and it should be adjustable for rake. Unfortunately, few home microcomputers are designed with the required low profile. Sinclair's ZX series, the Oric-1 and the Jupiter Ace are exceptions, but they all have even greater problems with their keyboards because of their use of either multi-layer membranes or moulded rubber sheets in place of sprung keys. Multi-layer membranes have no 'feel' whatsoever, and in the case of the ZX80 and 81, are spaced in such a way as to defy anyone to touch-type on them. The combination of these factors makes entering long programs an exhausting task. The Oric-1 and Spectrum attempt to circumvent this problem by producing an audible signal that a key has been depressed sufficiently to make contact. But that is hardly an adequate compensation. There are a number of companies supplying alternative keyboards — full size, with sprung keys — for the Sinclair computers, but the well designed examples are expensive. They also maintain the single key entry convention devised by Sinclair to speed operation in BASIC, which is a constant source of irritation for even a semi-skilled typist.

The ideal layout of a keyboard requires the rows of keys, as viewed from the side, to be arranged as if to form part of the circumference of a drum. This would minimise the directional movement of the typist's fingers. The only home computers that fit this specification are: the BBC Micro, the Commodore 64 (as well as the later Vic-20s), and the Apple II.

MARTIN BURKE

The Shorthand Machine
Where there is a need to record speech, and the stenographer has no means of slowing the speaker down, a device known as a Palantype is often employed. Shorthand machines of this type use a shorthand version of the phonetic spelling

The layout of the keyboard itself has long been a major bone of contention with designers. When typewriters first became available in the 19th century, there were as many different keyboard layouts as there were manufacturers, but in general the most frequently used character keys were grouped together at the centre of the keyboard. When the 'typebasket' was introduced, in the 1870's, manufacturers discovered that even quite slow typists could cause the type bars to jam against each other. The problem occurred most frequently with words such as 'ten', where the commonly used letters in the English language (which were conveniently placed next to each other on the keyboard) were used in rapid succession. The solution adopted was to move those letters most often found adjacent to each other in words, further apart in the typebasket — hence the now standard QWERTY keyboard, designed by Scholes and Gliden in the United States. There is no reason at all why an electronic keyboard should be constrained by this layout except to maintain a standard approach — an interesting example of a *de facto* global standard becoming undesirable and yet impossible to change.

However, some efforts to develop alternative keyboards have been made. In 1977, Mrs Lillian G. Malt employed the flexibility inherent in electronics hardware to produce a keyboard shaped to fit the hand, which is considerably less tiring to use than the standard design. It is also much quicker in operation — reports of 300 and more words per minute are commonplace. Unfortunately, it has not succeeded in breaking the QWERTY stranglehold on keyboard layout.

One very useful feature that this keyboard (called the Maltron) shares with many microcomputers is detachability. Most home computers do not have built-in monitors and are themselves small enough to be moved around, but this is not the case with many microcomputers designed for office use. Increasingly, keyboards are being designed to be as slim as possible and are attached to the microcomputer by an umbilical cord. IBM's PC Junior has gone one step further: the communications link between the keyboard and the microcomputer is similar to television and video recorder remote controls, and works by means of infra-red light.

Because ergonomics is not a totally objective science — it is the study of how workers *relate* to their working environment, and that relationship tends to change from time to time — it is not possible to give hard and fast rules. The keynote is long-term comfort.This requires the arrangement of tools and equipment so that all your energies can be devoted to the task in hand, without it being necessary to change position constantly, and without becoming unduly tired.

There are several further things that the home computer user can experiment with in order to improve his working environment. When we discussed Apple's Lisa (see page 261), we noted that there were alternatives to the keyboard when

Key Strokes
Before electronic keyboards were devised, each key on the typewriter had to be physically connected to the tiny character-shaped casting that made the impression on the paper. This imposed constraints on the layout of the keyboard, for it was essential to keep commonly used keys separated so that the bars that carried the characters would not clash. Though this is no longer necessary, we still retain the familiar QWERTY layout. Keyboards such as the Maltron, which places keys according to their frequency of use, have not proved popular

working with menu-driven software. You might care to attempt an inexpensive version of this using a joystick or trackball, and gauge for yourself the benefits. Of course, you will need to write some small programs to work with, but by using PEEK and POKE within the confines of screen memory this is not a difficult task.

Alternatively, if your computer allows you to re-specify the value for any particular key, you might care to rearrange the keyboard, sticking labels over the keys to indicate their new values. In this case it is perhaps easiest to PEEK the value of the eight bytes that make up the character into an array with eight variables, change the values within the array, and then POKE them back again. You could POKE the eight bytes that make up the character straight into the space allocated for the character that you wish to replace, but if you use this method remember to save the first set of values in a temporary array and then move each character to its new position in order. Save the program that performs this operation on cassette, because when you switch your machine off (or reset it), the value of each character will revert to the original!

Finally, if your interests extend to simple carpentry, you might consider constructing a purpose-built workstation, with the keyboard recessed into the worktop and the television set or monitor conveniently angled. Commercial versions of the workstation usually provide additional space for mass storage (disk or cassette drives) on shelves located under the worktop, and allow all the leads to be hidden away. Ergonomics is basically applied common sense, but a little thought will be repaid by a significant reduction in backache and eyestrain.

Future Alternatives
Many computer designers would dispense with keyboards completely if they could. Newer types of microcomputer, with larger memory and faster processing speeds, allow other devices such as joysticks, trackballs and mice to be used instead, with appropriate software

KEVIN JONES

Track Record

The function of the Disk Operating System (DOS) is to keep tabs on where everything is kept on the disk. Without a DOS, programming would be very hard work

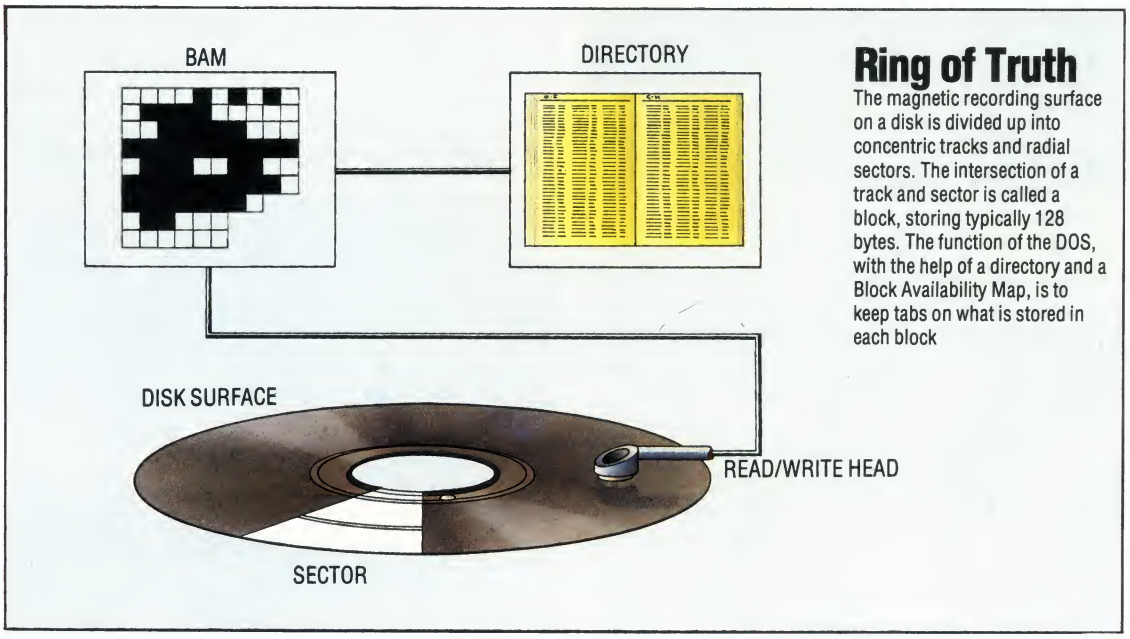
Before a computer is able to run any kind of applications program, it first needs its own internal set of programs to manage the various parts of its system, and to make sense of the instructions that comprise the user's program. This internal set of programs is called the Operating System (OS), and on most home computers this resides permanently inside the computer in the form of ROM memory. Generally, we are totally unaware that the Operating System is functioning, which is why we refer to it as being 'transparent in operation'.

If your system includes a disk drive then a large part of that OS will be concerned with the various disk operations. We call that set of routines the Disk Operating System, or DOS. You might see those three letters used in the names of proprietary products — Microsoft's operating system, for example, is called MSDOS. A DOS will typically come in one of three forms. It may comprise part of the ROM inside the computer. An example of this is the Sinclair Spectrum, which has the commands for operating the Microdrive (not really a disk, of course, but similar in operation) built in.

Another type stores the DOS in ROM within the disk unit itself. This is only applicable when the disk is an 'intelligent' device (such as the Commodore Disk Unit), meaning that it incorporates its own microprocessor ROM and RAM. These are more expensive to manufacture, but offer considerable advantages over 'non-intelligent' disk units. For instance, they don't eat up valuable user memory, and can be left to execute a complex disk operation while the computer itself continues with the applications program.

Thirdly, the DOS may reside inside the computer RAM. This technique is increasingly popular in business systems, in which the disk drives are built into the computer, and there is plenty of RAM available (say, more than 128 Kbytes as standard). For the manufacturer, this has the advantage of eliminating the need to create a completely new set of ROMs every time there is a minor modification to the DOS, and the user benefits from a choice of one of a number of proprietary operating systems that will run on the same hardware.

But how does the DOS get into RAM in the first place? This question immediately arises when the system is switched on. The DOS needs to be transferred from the disk into RAM, but if there is no DOS in the computer to tell it how to control the disk, how can it load something into RAM? A program cannot 'pull itself into RAM by its own bootstraps', so the computer has to have a tiny program built into ROM, which it executes whenever the machine is switched on. This program is called the 'bootstrap' (from the analogy above) and is itself a very simple form of DOS. The bootstrap's job is simply to find the



Ring of Truth

The magnetic recording surface on a disk is divided up into concentric tracks and radial sectors. The intersection of a track and sector is called a block, storing typically 128 bytes. The function of the DOS, with the help of a directory and a Block Availability Map, is to keep tabs on what is stored in each block

KEVIN JONES

main DOS on the disk, and transfer it byte by byte into RAM, whereupon that DOS can take over and perform some far more sophisticated functions. This process of switching the computer on, then waiting for the DOS to take over, is called 'booting-up'. When it is finished, a greeting is printed on the screen with a prompt to indicate that the computer is ready for a command from the user.

Whichever form the DOS in a system takes, its main function is looking after the locations of the contents of the disk. You may remember that a disk (see page 114) is divided into concentric rings, called tracks, which are in turn divided into sectors; and the intersection of a track and sector is called a block. A block can typically hold 128 bytes of information, and is the smallest unit that the disk can read or write at a time. One of the main reasons for having a DOS is to enable the computer to remember the exact location of everything on the disk. This task is more awesome than it sounds. Let's suppose our disk drive has a capacity of 320 Kbytes — enough to store 20 programs of 16 Kbytes each. With each block holding 128 bytes, loading one of those programs without the benefit of a DOS would require you to specify 128 different blocks, each with its own track and sector number!

Local Directory

Filename	Type	Location (Track-Sector)
Invaders	Prog	20-1,20-7,20-2...
Temperat	Prog	25-11,26-5,26-12...
Budget	Prog	23-12,24-3,24-9...
Budgetdat	Data	27-1,27-7,27-2...

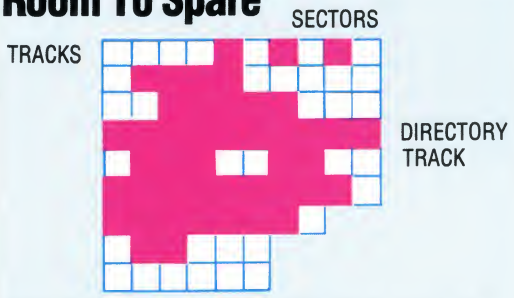
The directory on a disk typically occupies the centre track. It contains a list of the filenames, file types (program, data, and perhaps other categories) and track and sector numbers where the file is stored

In order to perform this function, the DOS keeps a disk directory. This is usually located in the middle track of the disk because it has to be referenced frequently, and this minimises the distance the read/write head has to move. The speed of operation of a disk is far more dependent on the time taken to move the head from track to track than on the speed at which the disk spins.

The directory is a list of all the files (which may be programs or files of data) currently on the disk, with details of the file name, file type, and a list of the blocks (each specified by track and sector) where that file is stored. There may be some other entries, such as the date when a back-up copy of the file was last made, or a list of the users who can access a particular file.

When a new file is to be stored, the DOS must first look up something called the Free Sector List or the Block Availability Map (BAM). This has a single bit corresponding to every block on the disk, and as a block is used the value of its bit is changed from zero to one. Some home computers with disk drives feature a utility program that displays the BAM on the screen, and you can watch the entries being made as you save a program. When a file is erased, the DOS doesn't bother to wipe clean all the blocks used in that file; it simply changes the entries in the BAM to indicate that the contents of those blocks are now unwanted.

Room To Spare



Before the DOS can store a new file and make an entry in the directory, it must first consult the Block Availability Map (BAM) or Free Sector List. This is a section of memory in which each bit corresponds to a block on the disk. A binary 1 indicates that the block is in use, 0 that it is free (we've shown it as solid or empty squares). Notice that the innermost tracks (at the bottom of the map) have fewer sectors than the others, because they are so much shorter

Another feature of this system is that files are not stored, as would be expected, in consecutive neighbouring blocks. Suppose, for example, that a track consists of 12 sectors, numbered 1 to 12 clockwise. The first 128 bytes of a program might be found in sector 1, the second in sector 7, the third in sector 2 and so on. This is because there is a small time lapse while a block's contents are transferred to the memory buffer used to write each block. If the DOS had to write consecutive sectors, it would have to wait for one complete revolution of the disk between each write — thus slowing the system down. Furthermore, a disk that has been in use for some time, with files that change in length each day, will end up with a BAM looking like a piece of Gruyère cheese, and new files will have to be fitted into the holes.

CHRIS STEVENS

I.Q. Test
Some disk drives contain their own microprocessor and RAM. These are called 'intelligent' drives, and the DOS is incorporated in the form of ROM. Where 'non-intelligent' drives are used, the DOS is stored inside the computer

A Disk Operating System has many other functions, including formatting new disks (marking out the tracks and sectors on a blank disk and creating an empty directory), making back-up copies, and 'tidying-up' full disks. More sophisticated versions include a variety of data handling structures (see page 204).

Outer Limits

The ZX81 is still the cheapest computer available. But with the right add-ons it can be expanded into a very sophisticated machine

Sinclair's ZX81 offers the best value for money of any microcomputer on the market today, even in its basic form. But there are a surprising number of add-on units available which can turn it into a remarkably sophisticated microcomputer system. These include high resolution colour graphics, speech synthesis and communications capabilities. Of course, the computer itself has some deficiencies, but these can be overcome by the addition of a variety of readily available items such as professional standard keyboards, extra Random Access Memory and programmable joystick controllers.

RAM Pack
In standard form the ZX81 has only one Kbyte of RAM, and 123 bytes of that are taken up by system variables. Consequently, memory expansion is perhaps the first requirement of a new owner. Sinclair's own memory upgrade shown here comes in only one form — 16 Kbytes — but alternatives such as the Cheetah version offer as much as 64 Kbytes,

Also Available...
In addition to the units shown here, there are other devices available to enhance the ZX81's performance. A colour card, for example, will provide up to 16 colours on the TV display, and a sound generator will give three programmable 'voices'. Bi-directional ports can support up to 16 input/output devices at once. Far from being a small and unsophisticated home computer best suited for playing games and learning the rudiments of BASIC programming, Sinclair's ZX81 can be expanded to meet the full potential of its Z80 microprocessor

Acoustic Couplers
Modulator/demodulators come in two forms: direct connection modems, which require an additional jackplug connector into the telephone system, and acoustic couplers such as the Micro-Myte 60, shown here, that use the telephone handset itself.

Direct-connect modems, which are generally rather more expensive, generate and recognise electronic signals that represent the 0s and 1s of the information being received or transmitted. Acoustic couplers, which may be battery powered, translate the 0s and 1s into audible tones for transmission over the telephone network, and perform the same process in reverse to receive information

Forth ROM
Sinclair ZX microcomputers use a rather idiosyncratic version of BASIC, and while it is not possible to install a different dialect, one can change the language completely — to FORTH, for example. There are two ways of doing this: either by loading the new language into RAM from cassette, which means that the computer will revert to BASIC each time it is switched on or reset; or by swopping the BASIC ROM for another. This FORTH-in-ROM from David Husband goes further than most — it allows ten or more programs to run on the computer simultaneously. This facility can only be fully exploited in control applications, where several devices must be programmed independently

MARCUS WILSON-SMITH

Speech Synthesis
Another interesting extra from Cheetah is the Sweet Talker speech synthesis unit. Sweet Talker uses an allophonic system, and hence is much less complex to program than units that work in phonemes (allophones are groups of like-sounding phonemes). There are other similar units available for ZX81 and many other home microcomputers

Hebot
The Hebot turtle, available either ready-made or in kit form, is one of the more sophisticated of the dedicated floor robots. It comes complete with software to drive it, and there are a variety of extras available such as photosensors, which can be used in conjunction with reflecting tape stuck down onto the floor, to make the robot follow a predetermined path

Keyboards
The multi-layer membrane keyboard is perhaps the ZX81's least satisfactory feature, so it is hardly surprising that a number of companies offer alternative full-size keyboards with conventional sprung keys. The Mapsoft ZX81 keyboard, shown here, is available from Maplin Electronics either as a kit or ready-built.

In addition to the normal character set, the Mapsoft keyboard provides three extra keys. For less than £30 it is a very useful addition to any ZX81, though similar products can cost more than twice as much. Another approach is a stick-on keyboard the same size as the ZX81's own, which is used in conjunction with the original. It makes locating a particular key rather easier, but has no other effect

Joystick Controller And Joystick
Given that many of the ZX81s sold must be used for games playing, it is perhaps rather strange that Sinclair has not produced its own joysticks and controllers. However, a wide variety are available, and these are either non-programmable, which specify for you which key strokes will be made by the joystick, or programmable, where the user decides which keys will be simulated. The model shown here, from AGF Hardware, is programmed by moving connecting cables around. Others are programmed through the computer. This one will accept any switch-type joystick, and also a trackball

ZX Printer
Sinclair's own ZX Printer uses aluminised paper that is sensitive to electricity. Instead of printing in a conventional manner, the print head removes the aluminised coating to reveal a darker surface beneath. While reasonably fast in operation, the limited paper type and width are major problems. It is possible to use a normal printer, however, via an interface card. Cards are available to support RS232 and Centronics interfaces

Top Gear

By paying careful attention to variables and program structure, you can speed up the operation of almost any Basic program

BASIC is, despite what its critics say, a versatile language and a powerful educational aid. You can write any program in BASIC, provided your machine has enough memory and the execution time is not important. However, because BASIC is usually interpreted rather than compiled (see page 184), it can be painfully slow in executing programs — especially those that require the same instruction to be translated and executed repeatedly.

Sorting, for example, is a highly repetitive process: the procedure is carried out within a loop, and there are smaller loops nested inside the main loop (see page 286). If 100 items are to be sorted, the program may make between 2,500 and 5,000 iterations of the loop. A BASIC sort will always be slow, but the way the code is written can make a significant difference to the speed of execution. If an instruction is to be repeated 5,000 times, and if coding it properly can save one hundredth of a second of execution time for each repetition, then there will be a total saving of 50 seconds — a considerable improvement for the user.

To observe the difference that good and bad coding can make, you will need a timing mechanism and a 'testbed' program. If you own a Commodore computer, you can use the system clock, with the associated variables TI$ and TI, as part of the testbed program. If your computer doesn't have an accessible clock, you'll have to use a stopwatch to time the code in execution. It is also a good idea to make your program 'beep' at you when it starts and finishes, so that you'll know when it's operating.

The testbed program looks like this:

```
1000 L=500
2000 PRINT "***GO***":REM "BEEP"
       instructions here
2100 TI$="000000"
2200 FOR K=1 TO L
.....
2900 NEXT K:T9=TI
2950 REM "BEEP" instructions here
3000 PRINT "******STOP******"
3100 PRINT "That took    "; (T9/60); "    seconds"
```

Lines 2100 and 3100 are for Commodore users. For other machines, delete or replace them with appropriate code. The space between lines 2200 and 2900 is where we will put the code to be timed. Notice that the timings will refer to L repetitions where L is the limit of the loop. Testing only one execution of a piece of code would be very inaccurate because the system clock measures only in 60ths of a second, and there is a timing overhead imposed by the code of the testbed program as well.

Here are some general rules for writing efficient BASIC, roughly in order of importance:

1. Avoid all arithmetic in loops.

Exponentiation (x^3, meaning 'x raised to the power of 3', for example), and mathematical functions (cos(y), meaning 'the cosine of the angle y', for example) are particularly slow. Multiplication and division are slower processes than addition and subtraction, but even the quickest of these operations (addition) is relatively slow.

In the testbed program insert these lines:

```
900 Z=1.1
2300 X=Z↑3
```

and run it. On our test machine 500 repetitions took 27.95 seconds. Now replace line 2300 with:

```
2300 X=Z*Z*Z
```

and run it. This took 3.55 seconds — a dramatic difference!

Further investigation will reveal the level of exponentiation at which it becomes worthwhile replacing repeated multiplication by the exponentiation function. On our computer this was at the 18th power (when X=Z↑18). Remember, however, that to calculate $Z^{2.3}$, for example, repeated multiplication would be useless, whereas the exponentiation function (↑) works for all real numbers, including negative ones.

Use the testbed program to see how long the other arithmetic processes take, and compare alternatives. Is it quicker to divide a number by 2, or multiply it by 0.5, for example?

2. Use variables rather than numerical constants.

Every time a numerical constant (7,280 for example) occurs in a BASIC instruction, time is spent translating the number into usable form. Try this line:

```
2300 X=X+7280
```

On our machine that took 4.63 seconds to execute 500 repetitions, whereas:

```
900 C=7280
2300 X=X+C
```

took 2.75 seconds to do the same number of repetitions.

3. If you must use the GOTO statement, jump forward in your program rather than back. If you must jump back, however, jump to the start of the program rather than back a few lines.

The same is true for GOSUB. On meeting a GOTO or GOSUB instruction the BASIC interpreter compares the target line number with the current line number. If the target is greater than the current, the interpreter simply searches forward, line by line, until it is found. But if the target is less than the current, then the search always begins from the very first line of the program. This means that it may be more efficient to place subroutines and frequently used sections at either end of a program. Add 56 REM lines at the start of the program, to make it up to typical length, and try:

```
2300 GOTO 2400
2400 GOTO 2500
2500 GOTO 2900
```

This took 2.33 seconds for 500 repetitions, whereas:

```
2300 GOTO 2500
2400 GOTO 2900
2500 GOTO 2400
```

took 4.85 seconds.

4. Initialise all variables in order of access frequency.

Variable names are stored by the interpreter in a symbol table in the order in which they first appear in a program. The later a variable occurs in the table, the longer it takes to find it and access its contents. For the same reason you should avoid using a new variable in a program where you can resort to one previously used by the program but currently not in use.

If a variable is used inside nested loops — as is common in sorting — that variable is accessed frequently, so initialise it at the start of the program before any other variable, with a dummy value if need be:

```
1000 L=500:C=7280:X=0:Z=1.1
2300 A=0
```

took 2.2 seconds for 500 repetitions, whereas:

```
1000 A=0:L=500:C=7280:X=0:Z=1.1
2300 A=0
```

took 2.06 seconds.

5. Avoid using strings.

String operations use up memory in ways that arithmetic does not, and a system program called the Garbage Collector may have to be called every now and again by the interpreter to tidy up the contents of string memory. This procedure can take a lot of time.

A general demonstration of this is difficult to write because computers vary so much in their memory management: you have to fill up most of the user memory with data — a large numeric array will do — then perform string manipulations that will cause the Garbage Collector to be called. On our machine we entered:

```
40 POKE 52,32:POKE 56,32:CLR
```

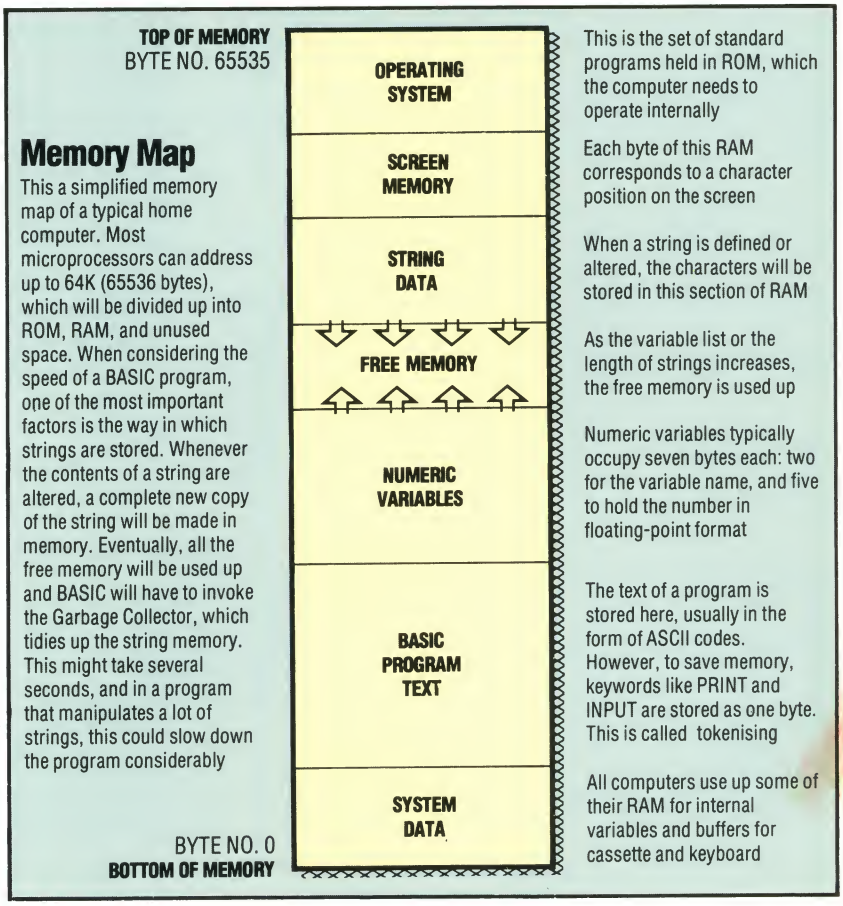
to reduce severely the amount of memory available to BASIC programs, and then entered:

```
1000 L=500:DIMT$(L)
1100 FOR K=1 TO L
1200 T$(K)="A"+"B"
1300 PRINT K
1400 NEXT K
```

which uses up a lot of string memory and provides a string array for later use. The PRINT statement is executed in every iteration, displaying the value of the loop counter. When we ran this version of the testbed program, the printing repeatedly paused as the Garbage Collector was called to rearrange memory. Sometimes the pause lasted more than three seconds. The program continues:

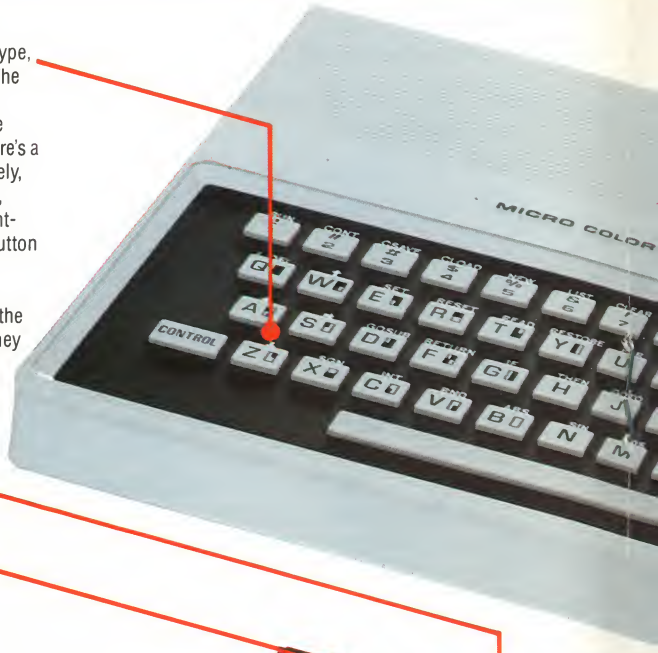
```
2300 A$=LEFT$(T$(L),1):B$=A$+RIGHT$(T$(L),1)
```

and this took 30.03 seconds for 500 repetitions. When we ran the same program with much more memory available, garbage collection was not visible, and the timed loop took 8.66 seconds.


TOP OF MEMORY
BYTE NO. 65535
Memory Map
This a simplified memory map of a typical home computer. Most microprocessors can address up to 64K (65536 bytes), which will be divided up into ROM, RAM, and unused space. When considering the speed of a BASIC program, one of the most important factors is the way in which strings are stored. Whenever the contents of a string are altered, a complete new copy of the string will be made in memory. Eventually, all the free memory will be used up and BASIC will have to invoke the Garbage Collector, which tidies up the string memory. This might take several seconds, and in a program that manipulates a lot of strings, this could slow down the program considerably
BYTE NO. 0
BOTTOM OF MEMORY
OPERATING SYSTEM
SCREEN MEMORY
STRING DATA
FREE MEMORY
NUMERIC VARIABLES
BASIC PROGRAM TEXT
SYSTEM DATA
This is the set of standard programs held in ROM, which the computer needs to operate internally
Each byte of this RAM corresponds to a character position on the screen
When a string is defined or altered, the characters will be stored in this section of RAM
As the variable list or the length of strings increases, the free memory is used up
Numeric variables typically occupy seven bytes each: two for the variable name, and five to hold the number in floating-point format
The text of a program is stored here, usually in the form of ASCII codes. However, to save memory, keywords like PRINT and INPUT are stored as one byte. This is called tokenising
All computers use up some of their RAM for internal variables and buffers for cassette and keyboard
LIZ DIXON

Tandy MC-10

Although clearly designed to be low in price this computer offers good colour, and many features of more expensive machines



Tandy MC-10 Keyboard
The keyboard is a button-type, but it's better than many. The keys are hard plastic with engraved legends that take longer to wear off, and there's a real space bar. Unfortunately, there's only one SHIFT key, which is placed on the right-hand side, and the large button on the left is the more conveniently positioned CONTROL key. The feel of the keys is comfortable, but they are not suitable for speed-typing

Reset Button
Because it's large and bright red, the reset button is much easier to find than on some other machines. When using the MC-10, take care not to bump the back of the machine too hard here

Cassette Interface
Remote control is provided through this five-pin DIN plug on pins 1 and 3. Signal input is on pin 4, output on 5, and the signal ground is on pin 2

RS232 Interface
This is also a DIN plug, but is constructed of four pins. Carrier detect is on pin 1, receive data on pin 2, ground is on pin 3 and transmit data on pin 4

System Bus
This is not explained in the manual, though it is obviously intended to be used with some unspecified expansion units. There are, however, enough lines to handle some complex devices

ROM
This is soldered firmly onto the board, and so is not likely to be replaced with upgrades or alternatives. The Microsoft BASIC is stored on the 8 Kbytes of ROM

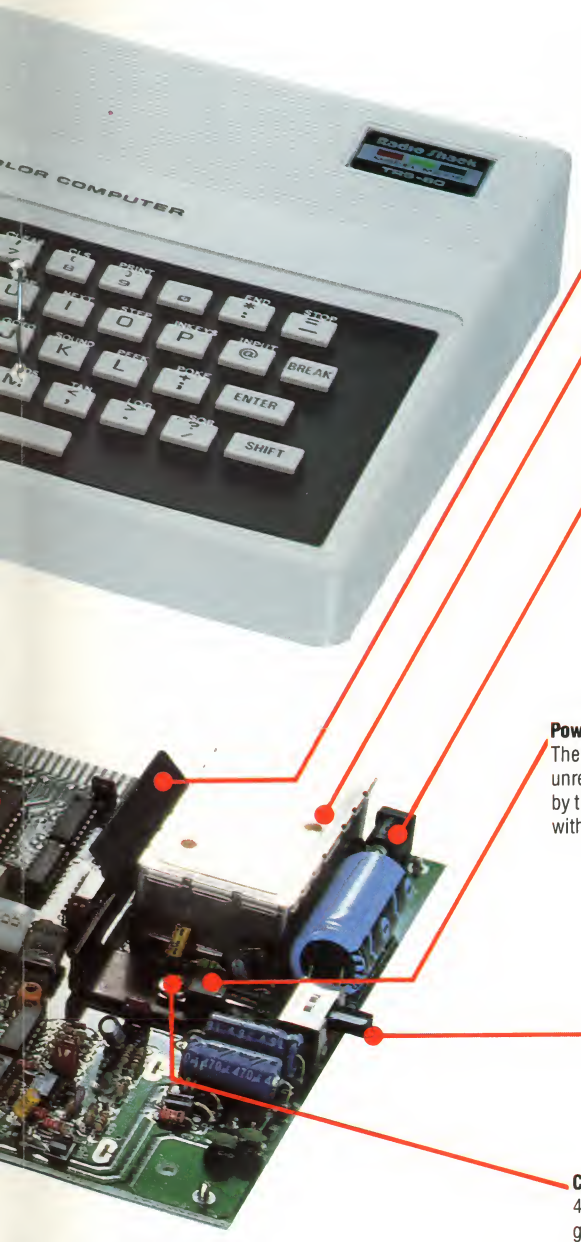
CPU
Unusually, the Tandy MC-10 uses a 6803 processor, rather than one of the more popular types. This processor is a member of one of the older families, and isn't as well-known as the 6502 or Z80. However, it's a useful 8-bit chip with a reasonable instruction set

Static RAM
The nominal 4 Kbytes of user RAM is held on these two 2 Kbytes x 8-bit static RAM chips, as are the screen RAM and some system variables

6847 VDP
In common with many other machines, the screen is controlled by a special chip, which in this case is the MC 6847 Video Display Processor. This chip is the same as that used in the Dragon 32, and (in theory at least) can be programmed for different screen formats. In practice, however, this is seldom done

The Tandy MC-10 is a compact little machine that achieves a lot using a few sophisticated chips. The keyboard is a button-type, though slightly larger than others of that kind, and possesses a proper space bar. Other features make the machine quite easy to use. Single-key BASIC keyword entry, for example, is achieved by holding the CONTROL key down while pressing the desired function key. The machine also defaults to 'all capitals' mode when it's switched on, and the lower case mode is a toggle — activated by pressing SHIFT 0, and de-activated by pressing the same keys again.

The screen display is smaller than that of most other home computers. There are only 16 lines of 32 characters, and only fairly low resolution graphics can be achieved. The display has other shortcomings as well, including rather limited colour facilities, although the quality of the colour is very good. Most surprisingly, it will not display lower case characters, which are recognised but shown as inverse upper case letters instead. Text can only be green on black or vice versa, and though the block-graphic symbols may be in any one of nine colours, either the letter or the



Heat Sink
The Triac power regulating transistor becomes very hot when it's running, and the heat is dissipated by this large piece of metal

TV Modulator
This converts the data stream produced by the video circuitry into a Channel 36 TV signal, but with no sound on the TV signal. This is the only screen output, and there is no monitor socket on the machine

Power Socket
This is a normal low voltage co-axial socket. In common with all machines of this type, the Tandy MC-10 takes its power from a small low voltage transformer plugged into a wall socket

Power Regulator
The transformed but unregulated power is stabilised by this large transistor, together with other nearby components

Power Switch
Since the MC-10 has a reset button, this does not need to be used as an alternative, as on some machines

Crystal
4.4 MHz is the frequency generated by the master clock, which is subdivided into slower pulses and used throughout the machine

CHRIS STEVENS

TANDY MC-10

PRICE
£49.95

SIZE
210 x 178 x 51mm

CPU
6803

CLOCK SPEED
4.4 MHz

MEMORY
8 Kbytes ROM
4 Kbytes RAM

VIDEO DISPLAY
16 lines of 32 characters, 9 colours with only background settable. 75 pre-defined characters

INTERFACES
RS232 serial, cassette

LANGUAGES SUPPLIED
BASIC

OTHER LANGUAGES AVAILABLE
NONE

COMES WITH
Operation and BASIC reference manuals, TV lead

KEYBOARD
48 button-style keys

DOCUMENTATION
Clear, competent and well-designed but rather lacking in technical information. The only major failing is the absence of an index. A quick-reference card is included, which gives enough details about the BASIC for an experienced person to start working the machine without delay

background must be black. Consequently, it's not possible to produce a blue shape on a red background, even in the graphics mode!

The sound function also has limitations. There is only one channel available, which allows minimal variations in pitch and duration only. Input/output facilities are to cassette (including remote control), television and an RS232 serial port. The serial port can be used as a data transfer line to and from other computers or, alternatively, to drive a printer. It can also be used to create a network with other Tandy MC-10s.

Games do not seem to have been a high priority with the machine's designers, who provided nothing in the way of paddle or joystick ports, nor any of the special graphics and sound controller chips found in other machines more suited to games playing.

Some expansion possibilities are clearly intended for the future, however, since there is a rather mysterious system-bus ending in an edge connector, which is covered by a screwed-on plate. Apart from stating that 'this slot is reserved for future memory expansion kits', the manual says nothing else about it, and provides no clues as to what accessories will be available to plug into it.

The documentation for the MC-10 is typical of that provided for Tandy's other machines: a rather aloof style of writing with few breaks in a fairly solid text.

As a low-cost machine, it is worth considering, but when reading the specifications remember that while it may have a nominal four Kbytes of RAM, only 3,142 bytes are available to the user, since the screen-RAM and some system variables have to come out of this allocation.

Hot Rods

Two new kinds of joysticks appear to have no moving parts. One uses mercury switches, the other picks up electromagnetic signals from your body

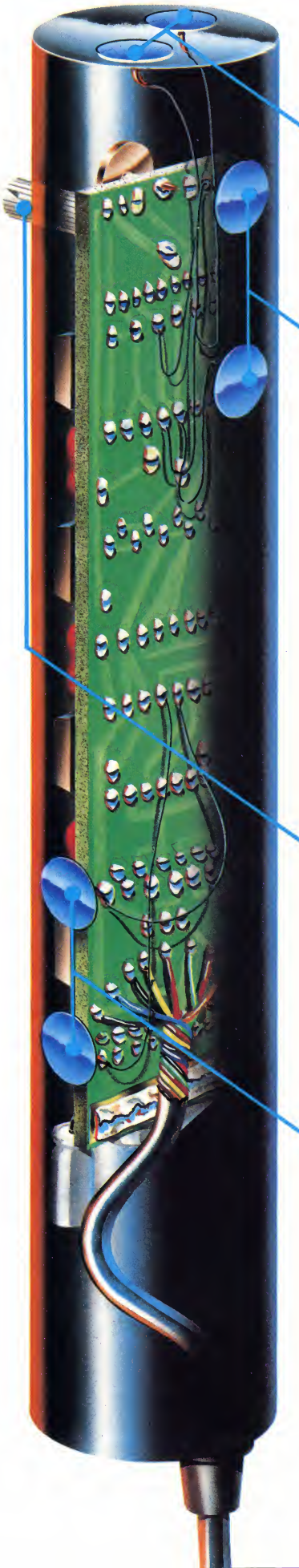
The personal computer industry is used to rapid technological developments, and these changes are not confined to the computers themselves — peripherals and add-ons are also subject to swift refinements. For instance, in the short time since we first discussed the mechanism of a joystick (see page 56), two completely new types have been marketed. The most recently developed joysticks have, in fact, almost entirely broken away from the conventional mechanical system described previously.

A device called Le Stik was the first analogue joystick to reject the usual signalling mechanisms. Le Stik consists of a contoured handgrip, fitted with a top-mounted fire button and a side-mounted pause control. Unlike other devices, which are mounted on base units, the joystick is simply held in the air and tipped from the vertical in the direction required, and the corresponding image on the screen moves accordingly.

The mechanism at the heart of Le Stik consists of four sealed tubes filled with mercury. As the joystick tilts away from the vertical the mercury flows in the chosen direction and makes one or more electrical contacts, just as though a switch

Hands On
The Trickstick relies on 'mains hum', which is the electromagnetic radiation given off by the ring main in every house. Your body acts as an aerial to mains hum, and the sensors on the stick pick up different levels of hum according to the pressure exerted by your fingers

Trickstick



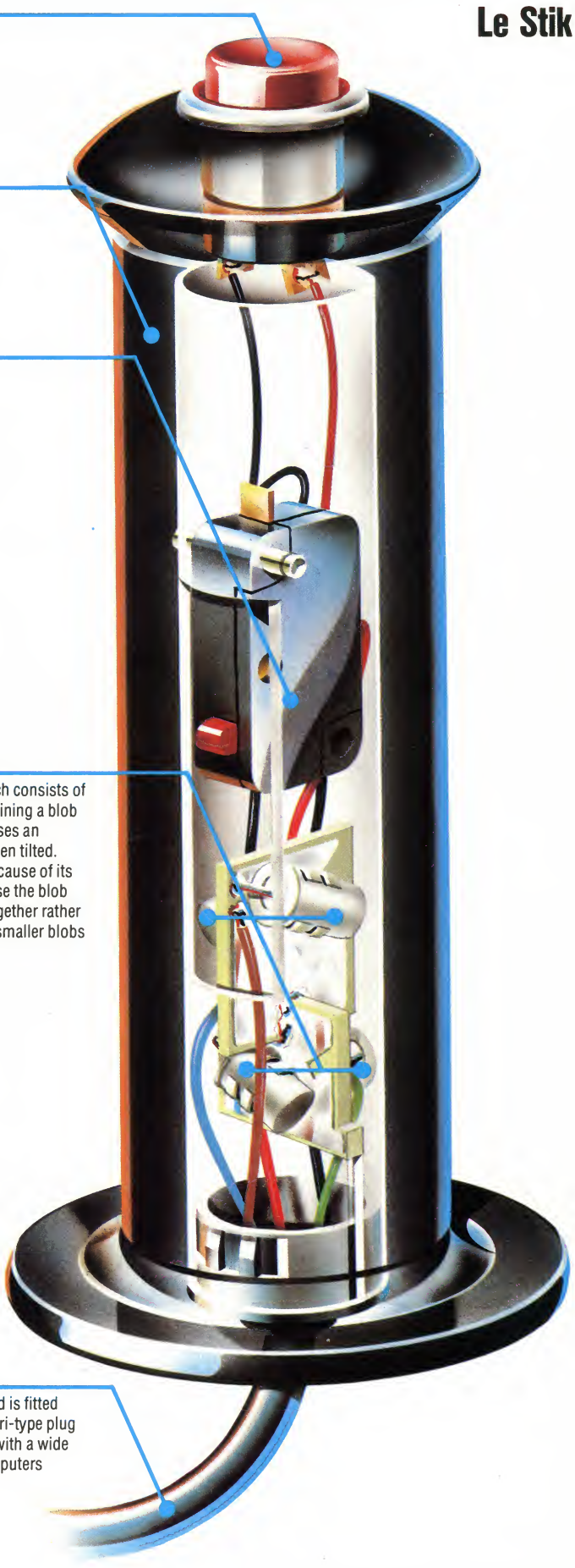
Horizontal Movement Controls
By rocking the thumb between these two pads, the forward and backward motion can be controlled

Vertical Movement Controls
The top pad controls upward motion, the lower one controls downward motion

Sensitivity Control
This allows the Trickstick to be adjusted to each individual player's efficiency as an aerial

Fire Buttons
Each generates an independent signal, so you could drop bombs with the bottom one and fire laser cannon with the other at the same time

Le Stik


Fire Button
The fire button is ideally placed for fast games action
Handgrip
This is one of the few available joysticks with a contoured handgrip suitable for both left- and right-handed users
Pause Button
The pause button, fitted into the handgrip, allows the player to take a break from the action between alien attacks, simply by squeezing the grip
Mercury Switches
Each mercury switch consists of a sealed tube containing a blob of mercury that closes an electrical circuit when tilted. Mercury is used because of its density, and because the blob will tend to stick together rather than break up into smaller blobs
Connecting Lead
The connecting lead is fitted with a standard Atari-type plug that is compatible with a wide range of home computers

had closed. Moving the handgrip back to a vertical position allows the mercury to flow back into the tubes, thus breaking the contact. The response of the system is considerably better than that of previous joysticks. Indeed it is often too sensitive, especially if the game being played is written for use with the conventional types of joystick.

The latest method of converting hand movements into signals that a computer can understand is used by East London Robotics' Trickstick. This joystick is unique in the electrical effect it employs: it uses the human body as an aerial to pick up mains hum (the harmless electromagnetic radiation emitted by the ring main in any room). Trickstick consists of a sealed tube in a plastic casing, which is held vertically in both hands. There are three pairs of touchpads set into the surface of the tube: one pair controls the forward and backward motion; another pair controls the up and down movement; and the remaining pair are the fire buttons.

The mains hum that the human body picks up is transmitted through these touchpads to sensitive circuitry, where the pulses are converted into signals that provide the computer with directional information. The signals can also be analysed to show how far the movement should be taken. The harder one presses a pad, the stronger the signal and the more rapid the output to the computer. In this way, the Trickstick combines the proportional control of the analogue joystick with the fast direct digital control of a switch-based unit. Because different people will affect the circuits in different ways, the Trickstick has to be adjusted for individual sensitivity. This is done by means of a knob mounted in one end of the device.

The idea is certainly intriguing, and the manufacturers have applied for a patent on the technique. However, the reliability and performance of the device have yet to be proven.

Sound Systems

A second look at the Vic-20's sound capabilities

Last time we looked at the Vic-20 in the Sound And Light series we learnt how the machine's three oscillators can be controlled by POKEing memory locations; how to set the volume levels; and how to control the duration of a note. We investigated how the duration of the notes and the pauses between them can be determined by the use of FOR...NEXT loops or, more efficiently, by using the Vic-20's clock to count in jiffys (60ths of a second). Management of these three musical elements — frequency, volume and timing — enables you to build simple tunes on the Vic-20 and produce useful sound effects.

Playing Tunes

To construct a tune you must first assemble the required notes. These could be, for example, the notes of the first line of 'Oh, I do like to be beside the seaside'. In the correct order these can be selected as:

D# E F D# C A# G# G G# D# D#

Using the techniques described on page 284, the duration of the notes and pauses can be set by using the TI facility. Our tune can therefore be played by RUNning the following program (notice the use of variables to simplify the selection of POKEs):

```
10 V = 36878
20 FOR I = 1 TO 11
30 READ N: REM *NOTE*
40 POKE V,7:P=TI: REM *VOL ON*
50 IF TI-P < 15 THEN 50: REM *PAUSE*
60 POKE V-3,N:D=TI: REM *PLAY NOTE*
70 IF TI-D < 20 THEN 70: REM *DURATION*
80 POKE V-3,0: REM *STOP NOTE*
90 NEXT I
100 DATA 203, 207, 209, 203: REM *NOTE VALUES*
110 DATA 195, 187, 179, 175
120 DATA 179, 203, 203
130 POKE V,0: REM *VOL OFF*
140 END
```

The Light Program

First steps with the BBC's sophisticated graphics

The BBC Micro is one of the most popular home computers in Britain. Truly stunning graphics effects can be simply achieved in a few lines of BASIC, and the speed at which displays are produced on the screen using BASIC is also impressive.

There are several high resolution commands in BBC BASIC, including instructions to draw straight lines, plot points, and plot and fill triangles. This last function is used to colour in shapes as a series of small triangles as there is no PAINT-type command available. The BBC Micro also lacks a BASIC command to draw circles and ellipses, and has no sprite programming capability. However, it does have several unusual and interesting features that the majority of its rivals do not possess. These include the ability to mix text and graphics on the screen, separately controllable text and graphics cursors, and access to the part of the machine operating system that controls screen display, from within a BASIC program. This is accomplished by the set of VDU or screen commands. Text and graphics 'windows' can also be defined on the screen, enabling the user to divide up the display into separate sections for graphics and text. Different colours can be defined for each window and each may also be cleared independently.

Display Modes

The BBC Micro has eight graphics modes, three of which support text displays only. There is a choice of 20, 40, or 80 characters across the screen, depending on which mode has been selected. Two, four or 16 colours are available, again depending on the mode selected, but a pleasing feature of the limited colour modes is that two or four colours to be used in that mode are not fixed and can be selected by the programmer from the 16 generally available.

MODE 7 is different from all the others in that the standard set of ASCII characters and associated codes are not used. Instead, the display is made up of Teletext characters. Normal graphics commands, such as PLOT and DRAW, do not work in MODE 7.

The following table shows the resolution and colour choices specified by selection of any mode:

Mode	Text	Graphics	Colours
0	80 × 32	640 × 256	2
1	40 × 32	320 × 256	4
2	20 × 32	160 × 256	16
3	80 × 25		2 (black & white)
4	40 × 32	320 × 256	2
5	20 × 32	160 × 256	4
6	40 × 25		2 (black & white)
7	40 × 25		Teletext

This program simply plays the notes in the correct sequence with equal durations and pauses. Consequently, the resulting tune is somewhat stilted. With experimentation you can construct more complex programs that provide different intervals and durations for individual notes.

Sound Effects

By using two or three oscillators it is possible to play simple chords. The program below plays the chord of D major (F#, A and D) starting with the F# on its own, and adding the A and D after set delays of one second each. The chord then continues for a further two seconds.

```
10 POKE 36878,7
20 POKE 36874,233:D=TI
30 IF TI-D < 60 THEN 30
40 POKE 36875,219:D=TI
50 IF TI-D < 60 THEN 50
60 POKE 36875,147:D=TI
70 IF TI-D < 120 THEN 70
80 POKE 36878,0: POKE 36874,0
90 POKE 36875,0: POKE 36876,0
100 END
```

A lot can be done, however, to make the tone of these sounds more interesting. For instance, the volume can be varied over the duration of a note — rising and falling according to a variable. For example:

```
100 V = 36878
110 FOR I= 1 TO 12
120 POKE V,I
130 NEXT I
140 POKE V,0
```

This causes the volume to rise in steps of 1 to a peak of 12, where the total range is from 0 (off) to 15 (loud). Volume can be 'pulsed' by alternating a high and low volume setting, as well. The frequency can be similarly varied to 'bend' a note by changing line 120 above to:

```
POKE V-3,203+I
```

It is also worth trying different combinations of noise, oscillator frequencies and volumes. This can often result in a more pleasing tone. Whether making music or adding sound effects to games, the aim in computing is to reduce boredom by avoiding the constant repetition of monotonous notes.

We have shown how the simple sound facilities on the Vic-20 can be manipulated to produce interesting tones and note sequences. The main problem is the lack of sound commands, which involves the use of complex BASIC statements to carry out relatively simple tasks. This results in long program routines that prevent the BASIC interpreter from processing the code in between notes quickly enough. The only simple way to avoid this problem is to invest in one of the many commercial software packages that supply extra commands for music programming. Commodore's Super Expander cartridge provides a useful range of sound commands, as well as a facility for storing tunes written with the aid of the cartridge. However, if you require more than rudimentary sound or music facilities from a home computer it would be necessary to investigate other models, such as the BBC Micro, the Commodore 64, the Dragon 32 or the Oric-1.

The high resolution screen is defined with its origin in the bottom left-hand corner of the screen, regardless of the mode selected. Vertical values range from 0 to 1023, and horizontal values range from 0 to 1279. This consistent method of mapping the screen becomes very convenient when you decide to change the display from one mode to another. Incidentally, if the mode of display is changed during the course of a program then the screen is automatically cleared.

Background, text and graphics colours are set using the COLOUR and GCOL commands. The BBC Micro uses the interesting idea of logical and actual colours to allow the user to select a limited set of colours from the 16 allowed. To illustrate this it is best to use the example of using colour in MODE 0 where only two colours can be specified. Two possible foreground colours are given the logical colour numbers 0 and 1, and unless the computer is instructed to do otherwise, it takes 0 as black and 1 as white. The COLOUR command selects the text foreground colour. COLOUR 1 would select logical colour number 1 as the text colour, but it is possible to reset the logical text colour using one of the VDU commands. VDU19 defines the logical colour. To set logical colour 1 to green (actual colour number 2) the following command is needed:

```
VDU19, 1, 2, 0, 0, 0,
```

The three noughts on the end have no significance and are there for future expansion of the system.

The GCOL command has two numbers associated with it. The second number is the logical colour number for graphics display, the first relates to the way in which that colour is used on the screen. For the command GCOL a,b values of a can range from 0 to 4 allowing the user to specify whether the point or line should be displayed in the logical foreground colour, whether it should be ANDed, ORed or exclusive ORed with the colour already present, or whether the original colour should be inverted.

In a future part of the Sound And Light course we will return to the BBC Micro and explain high resolution capabilities, defining characters, and look more closely at the set of VDU commands.

Changing Places

After looking at how to insert new records, we move on to ways of retrieving them. As anticipated, we first encounter the problem of finding an exact match

We ended the last instalment with an exercise for you to write a database-type program that allowed data to be entered into it. Let's look at some of the steps involved in entering a new record as a way of continuing our examination of what is involved in the INITIALISE stage of our main program. First, let's assume that there are the following fields and corresponding arrays:

FIELD	ARRAY
1 NAME field	NAMFLD$
2 MODIFIED NAME field	MODFLD$
3 STREET field	STRFLD$
4 TOWN field	TWNFLD$
5 COUNTY field	CNTFLD$
6 PHONE NUMBER field	TELFLD$
7 INDEX field	NDXFLD$

The meaning of most of these fields should be reasonably clear, with the possible exception of fields 2 and 7. Let's first consider the MODIFIED NAME field. When we initially looked at the problem of the data format for the name, we debated whether to have the name format tightly specified (rigid) or loosely specified (fuzzy) and we opted for the latter. Since the way a name can be entered is extremely variable, a rigid format would have made search and sort routines very difficult. To solve this we decided that all names would be converted to a standardised format: all letters converted to upper case, all non-alphabetic characters (such as spaces, full stops, apostrophes, etc.) removed and that there would be only a single space between the forename (if any) and the surname.

The need to standardise names like this arises because the sort and search routines have to have some way of comparing like with like. On the other hand, when we retrieve a name and address from the database, we want to have the data presented in the form it was originally entered. There are two ways of handling this problem: either each name filed is converted into standard form only when sorts and searches are taking place, or the name field can be converted into standard form and stored as a separate field so that sort and search routines can have instant access to standardised names.

There are advantages and disadvantages in both approaches. Converting the name fields temporarily when they are wanted by other routines saves memory space, since less data needs to be stored in the file. On the other hand, this procedure is extremely time-consuming. However, if a separate field is reserved for the standardised form of the name, the conversion will need to be performed only once for each record. And although extra memory is consumed, searches and sorts will be executed quicker.

The other field that may cause confusion is the INDEX field. This is really included as a spare field to allow for future expansion or modification of the database without the need for major rewriting of the program. Its inclusion introduces the topic of 'binding' — a term that means the fixing of data and processing relationships. All the fields or elements in each of the records are bound because they have the same index (the same element number or subscript in their respective arrays), and because all the fields in a record will be stored in a file together. This can make the addition of new data types or relationships at a later stage a difficult task, possibly involving the complete reorganisation of the file structure and a major re-writing of the program. The incorporation of the INDEX field at this stage will make future changes to the program much simpler.

Before attempting to add a new record to the database, we will make a few assumptions about the structure of the files. First, we will limit the number of records to 50 (even though this is really too small for a useful address book — we'll find out how to handle large amounts of data later). We will also assume that all the data has already been transferred — as part of the INITIALISE procedure — into arrays.

When a new record is added, it is simplest to add it to the end of the file (that is, to the first empty element in each array). There is a good chance that the new record will be out of order with the others, but that is a problem we can investigate later. The first thing to do, therefore, will be to find out how big the array is. Since this is a piece of information likely to be useful in many parts of the program, the best place to do it is in INITIALISE. This is a clear case of the need for a global variable (that is, a variable that can be used in any part of the program). We will call it SIZE. Another global variable likely to be useful is the index of the current record. Since no record will be current when the program is first run, assigning an initial value to CURR will have to wait until the program does something to the data. CURR can, however, be initialised to zero in the INITIALISATION procedure. Initialising a variable to zero is not strictly necessary in BASIC as this is done automatically. It is, however, good practice and

TONY LODGE

should always be done for local variables to prevent 'side effects' from the use of the same variable elsewhere in the program.

When the program is first run, various types of initialisation will take place and data will be loaded from disk or tape and transferred to string variables. The CHOOSE menu will then be presented. If the user chooses option 6 (to add a record to the file), the value of variable CHOICE returned will be 6, and this will call the sub-program ADDREC. ADDREC will assume that SIZE has already had a value assigned to it and so it can start prompting for inputs (note: this also assumes that INITIALISE has already correctly DIMensioned the necessary arrays).

Adding a new record also means that the file is now, potentially at least, out of order. Since a sort may take some time, it may not be necessary to sort the records after each addition has been made — that is a decision we shall defer for the moment. Instead, we will set a flag to indicate that a new record has been added.

We are now in a position to start making a tentative list of possible arrays, variables and flags that may be needed by the program.

ARRAYS

NAMFLD$	(name field)
MODFLD$	(modified name field)
TWNFLD$	(town field)
CNTFLD$	(county field)
TELFLD$	(telephone number field)
NDXFLD$	(index field)

VARIABLES

SIZE	(current size of file)
CURR	(index of current record)

FLAGS

RADD	(new record added)
SORT	(sorted since record modification)
SAVE	(save executed since record modification)
RMOD	(modification made since last save)

It is likely that in the course of developing the program a few more arrays will be needed. Certainly more variables will be needed. As for the flags, it is apparent that although others will be necessary, the four given above may not all be required. There will be no need either to save or sort the file (assuming it is already saved and sorted) unless a modification has taken place, so RMOD is possibly the only flag really needed. But if we do decide to use all four flags, the INITIALISATION sub-program should set them all to their appropriate values. As further practice in top-down program refinement, let's see how easy it is to code *ADDREC*.

I 4 (EXECUTE) 6 (ADDREC)

```
BEGIN
   Locate current size of file
   Prompt for inputs
   Assign inputs to ends of arrays
   Set RMOD flag
END
```

II 4 (EXECUTE) 6 (ADDREC)

```
BEGIN
   (size of file is SIZE)
   (prompt for inputs)
   Clear screen
   Print prompt message for first array(SIZE)
   Input data to array(SIZE)
   (prompt and input for all arrays)
   Set RMOD to 1
END
```

All this is straightforward and does not involve loops or other complicated structures. The next step can be direct coding into BASIC. The only points to note are that SIZE is a variable set during the execution of INITIALISE and does not need to be coded as part of this section.

III 4 (EXECUTE) 6 (ADDREC) BASIC CODE

```
CLS: REM OR USE PRINT CHR$(24) ETC TO CLEAR
   SCREEN
INPUT "ENTER NAME";NAMFLD$(SIZE)
INPUT "ENTER STREET";STRFLD$(SIZE)
INPUT "ENTER TOWN";TWNFLD$(SIZE)
INPUT "ENTER COUNTY";CNTFLD$(SIZE)
INPUT "ENTER TELEPHONE NUMBER";
   TELFLD$(SIZE)
LET RMOD=1
LET NDXFLD$=STR$(SIZE)
GOSUB *MODNAME*
RETURN
```

The third to last line sets the NDXFLD$ field to the value of SIZE (converted into a string by STR$), so that it can act as an index at a later stage. The subroutine *MODNAME*, called just before the end of the program, is none other than the program described in detail on page 254. A few slight changes will be needed to that program, but these are just details. This subroutine has the function of taking the ordinary (fuzzy) name input and converting it into a standard form. The output from this subroutine will be an element (SIZE) in an array called MODFLD$. All name searches and sorts can now be conducted on the elements in MODFLD$, and since the element will have the same index as the other fields in the record, it will be easy to display the name and address as they were originally entered. In other words, the search will be made on MODFLD$ but the display will come from NAMFLD$.

That's about all that's involved in adding a new record to the file, although we have not made allowances for any error checking, or provision for what would happen if there is no more space left in the array. Since all our programs are being written in modular form, modifications and improvements such as these can easily be made later without having to rewrite the whole program.

The sub-programs MODREC and DELREC (to modify and delete records respectively) are fairly similar to ADDREC, except that before they can be executed we have to locate the record we want to change. Consequently, both of these sub-

programs will start by calling FINDREC. This sub-program is based on a search routine similar to the one described on page 273. The chief difference this time is that (in all probability) no two data items will be identical, since few people have completely identical names.

There are two ways a search can be conducted. One is to search through an unordered pile. This makes the searches slower than they need to be. In the worst case, the routine might have to search through all of the data items before locating the item being searched for. Searching through an unordered pile does have the advantage, however, that sort routines are not required every time a record is added, deleted or modified.

If the data is ordered in some way — either numerically or alphabetically, for example — the program will have to search through only a small fraction of the items in the list. The longer the list is, the more efficient a binary search becomes compared with searching through an un-ordered pile. In fact, if there is enough data in the file to warrant it, the sorting of the records after a modification can be speeded up by conducting a preliminary search to locate the first and last occurrence in the array of the initial letter of the surname in the record involved.

Another way to speed up the sort routine might be to maintain a look-up table of the locations in the array of the first occurrence of each letter of the alphabet. This table, however, would need to be carefully maintained (updated) whenever any changes were made to the data.

The subject of searching and sorting is one of the largest areas in programming, and books have been devoted to it. We will not attempt to find the optimal solution for our address book program since this depends on a large number of factors, including the number of records in the file and whether or not disk drives are available.

A program in pseudo-language for a search through the elements in the MODFLD$ array is now given. The string variable KEY$ is the key for the search. The term 'key' here means the identifying group of characters used to specify which record (or records) is required.

```
Prompt for name to be searched
LET KEY$ = name (to be searched)
LET BTM = 1
LET SEARCHING = 0
LET TOP = SIZE
LOOP while (BTM < = TOP) AND (SEARCHING = 0)
   LET MID = INT ((BTM + TOP)/2)
   IF KEY$ = MODFLD$(MID)
      THEN
         PRINT NAMFLD$(MID)
         PRINT STRFLD$(MID)
         PRINT TWNFLD$(MID)
         PRINT CNTFLD$(MID)
         PRINT TELFLD$(MID)
         LET SEARCHING = 1
      ELSE
         IF KEY$ > MODFLD$(MID)
            THEN LET BTM = MID+1
            ELSE LET TOP = MID–1
         ENDIF
   ENDIF
ENDLOOP
IF SEARCHING = 0 THEN PRINT "RECORD NOT
   FOUND"
END
```

This piece of pseudo-language is closely based on the program used for searching football scores on page 275, but you will see that it does have a suitable output if the record cannot be found (the last PRINT statement), which will be executed only if the loop fails to locate an exact match between KEY$ and MODFLD$(MID).

Unfortunately, an exact match is rather unlikely, even if the name and telephone number you want is in the database. This is because the IF KEY$ = MODFLD$ statement is totally inflexible; it does not allow for the slightest difference between the character string input by the user in response to the prompt and the character string stored in MODFLD$(MID). In an ordinary address book, the eye scans down the page and is able to allow for all sorts of small differences in the actual representation of the record and what you are looking for. The computer cannot do this.

There are, however, ways of avoiding this, although they all involve extra programming effort and will take a little more time to run. The first improvement would be to check only the surname first, and for this reason it makes sense for the name stored in MODFLD$ to be in the form SURNAME (space) FORENAME. We developed a routine for reversing the order of a name earlier in the Basic Programming course (see 'Basic Flavours') and this can be incorporated as a subroutine within the ADDREC routine when the MODFLD$ field is created.

Having successfully located the first occurrence of the required surname, the FINDREC routine should then check the forename part of that element to see if it is identical to the name input (KEY$). If it is, there is no problem — the record has been located. If it is not, however, the problem starts to get complicated, and we have to plan our strategy carefully. We could, for example, search through all the forenames, and if an exact match is not found, start looking for an approximate match. The difficulty here is: what exactly constitutes an approximate match?

Instead of the "RECORD NOT FOUND" message in the program above, it might be better to give a message like "EXACT MATCH NOT FOUND, TRY FOR A CLOSE MATCH? (Y/N)?" What do the words 'close match' mean? Is Bobby a close match to Robert? How about Robrt? Both of these represent possible inputs in the FINDREC program. Let's try to define what we mean by a close match and then start to develop a program in BASIC to find the closest match to an input string.

Suppose the string in memory was ROBERT. Which of the following is the closer match: ROB or RBRT? The second gets four letters right out of six,

while the first gets only three out of six. On the other hand, the first has three letters in correct sequence, while the second has only two.

The choice is largely arbitrary. We will opt for giving priority to an exact match between KEY$ and a substring of the name in memory. If no exact match with a substring can be found, the program will try to get the largest number of common letters. Here's the program stated in terms of input and output:

INPUT
A character string
OUTPUT
The closest match to the input string

The following program, in a pseudo-language close to BASIC, will search through the strings in an array and examine the first 'n' letters in each, where 'n' is the number of letters in the key (KEY$). If there is no match, a message to that effect will be printed:

```
DIM ARRAY$(4)
FOR L = 1 TO 4
READ ARRAY$(L)
NEXT L
DATA "ROBERT", "RICHARD", "ROBIANA",
   "ROBERTA"
LET KEY$ = "RON"
LET LKEY = LEN(KEY$)
LET SEARCHING = 0
LOOP FOR INDEX = 1 TO 4
   IF KEY$ = LEFT$(ARRAY$(INDEX),LKEY)
      THEN PRINT "MATCH IS ";ARRAY$(INDEX)
         LET SEARCHING = INDEX
   ENDIF
ENDLOOP
IF SEARCHING = 0
   THEN PRINT KEY$; "IS NOT AN EXACT MATCH
         OF ANY"
      PRINT "FIRST ";LKEY; "CHARACTERS"
```

After this, the program could go on to look at groups of characters LKEY long, starting with the second character in each string. If none of these matches, groups starting with the third character could be searched, and so on. Finally, if none of the triplets of characters in the strings matches, the program could try to find which string had the largest number of letters in common with KEY$. This is left as an exercise for the reader.

We could in fact write pages on the subject of 'fuzzy' matching, and the different techniques employed in commercial database packages. Most offer the ability to search on the first few characters in the field, like the code we have just been developing. Others will retrieve a record if the specified sequence of characters appears anywhere in the field, or indeed anywhere in the record. A 'wildcard' facility is particularly useful, so that specifying: J?N would find JONES, or JANE but not JOHN. The most sophisticated form of fuzzy matching works phonetically, so that entering SMITH would also find SMYTHE.

Basic Flavours

ZX81 SPECTRUM

This is the listing of the program to reverse the order of Firstname and Surname, first published on page 136:

```
100 CLS
200 PRINT "ENTER NAME IN THE FORM"
300 PRINT "FIRSTNAME SURNAME"
400 PRINT "E.G. JILL THOMPSON"
500 INPUT "ENTER NAME";N$
600 GOSUB 9500
700 PRINT "NAME IN STANDARD FORM IS"
800 PRINT N$
1000 STOP
9500 REM S/R TO REVERSE NAME ORDER
9520 GOSUB 9600
9540 IF P=0 THEN RETURN
9560 LET N$=S$+",   "+F$
9580 RETURN
9600 REM S/R TO SLICE N$ AT A SPACE
9620 LET N=LEN (N$)
9630 LET P=0
9640 FOR K=1 TO N
9650 IF N$ (K)=" " THEN LET P=K
9655 IF N$ (K)=" " THEN LET K=N
9660 NEXT K
9670 IF P=0 THEN RETURN
9680 LET F$=N$( TO P-1)
9700 LET S$=N$(P+1 TO )
9720 RETURN
```

LEFT$
RIGHT$
MID$

On the Commodore 64, Vic-20, Oric-1, and Lynx, replace lines 9600 to 9720 of the Spectrum listing by these lines:

```
9600 REM S/R TO SLICE N$ AT A SPACE
9620 LET N=LEN (N$)
9630 LET P=0
9640 FOR K=1 TO N
9650 IF MID$(N$,K,1)="    " THEN LET P=K:
      LET K=N
9660 NEXT K
9670 IF P=0 THEN RETURN
9680 LET F$=LEFT$(N$,P-1)
9700 LET S$=RIGHT$(N$,N-P)
9720 RETURN
```

INSTR

On the Dragon 32 and the BBC Micro, replace lines 9600 to 9720 of the Spectrum listing by these lines:

```
9600 REM S/R TO SLICE N$ AT A SPACE
9620 LET N=LEN (N$)
9640 LET P=INSTR(N$, "  ")
9670 IF P=0 THEN RETURN
9680 LET F$=LEFT$(N$,P-1)
9700 LET S$=RIGHT$(N$,N-P)
9720 RETURN
```

As we have mentioned before, INSTR is a useful function, particularly when dealing with database-type applications such as this. If your machine has INSTR, then you may like to attempt a more sophisticated form of 'fuzzy' matching.

INPUT

On the BBC Micro, replace line 500 of the Spectrum listing by:

```
500 INPUT "ENTER NAME", N$
```

Konrad Zuse

COURTESY OF SIEMENS – MUSEUM, MUNICH

While von Neumann was doing his pioneering work in the USA, Zuse was achieving similar results in Germany

Inventions are often made simultaneously in different parts of the world from ideas that have been developed independently of each other. In the 1940's, while the first valve computer (ENIAC) was being developed in America, a German engineer, Konrad Zuse, was at work on a programmable calculator — arguably the world's first computer.

Zuse was born in Berlin on 22 June 1910. After attending the city's Technical University he worked as an aeronautical engineer for the Henschel Aircraft Company, developing wing design. The basic mathematical principles involved in strengthening aircraft wings to withstand the stresses of high-speed flying had been laid down in the 1920's. But the individual calculations needed for the production of each pair of wings required teams of people working with mechanical adding machines and slide rules. Zuse soon came to appreciate the need for a machine that could do this time-consuming work rapidly. Working with friends in his parents' flat in the evenings, he set about building a computer that could perform this task.

His first machine, the Z1, was a mechanical device that could perform the four elementary arithmetic operations, calculate square roots and convert decimal numbers to binary notation and vice versa. Although unaware of the achievements of Charles Babbage (see page 220), whose Difference Engine had been created to perform the laborious calculations needed for nautical tables, he had arrived at many similar conclusions and some that were far in advance. Zuse's major breakthrough was in recognising that a lever was a switch that could be put in one of two positions — on and off — and could therefore be used either as a means of storing data or as a control device.

Zuse pursued the idea of representing both data and instructions in binary form, and in 1941 he set out to build an electromagnetic computer, which he called the Z2. Involved in the war effort, the German government at first showed little interest in his invention. However, its military potential was eventually recognised and funds were provided for him to develop the new Z3. This was to be an electrical computer, with electrical wiring in place of the mechanical linkages that he had used in the earlier machines, and which allowed for a more compact and elegant design.

Zuse built the Z3 despite major handicaps. The Allied bombing of Berlin forced him to move his workshop several times. He was called up twice, only to be returned from the eastern front to continue his work. The wartime shortage of materials forced him to improvise by scavenging components from telephone switching gear and using old cinema film, punched with codes of eight holes per frame, in place of paper tape.

The Z3 could store 64 words, each of 22 bits in length. Information was input through a keyboard and the results displayed visually on an arrangement of lamps mounted on a board. Sadly, the Z3 was destroyed, along with Zuse's earlier computers, in the saturation bombing of Berlin in 1945.

One of the computers was adapted by the Henschel Aircraft Company to help in the construction of the HS-293 flying bomb. This was an unmanned plane that was launched from an airborne bomber and guided to its target by radio control.

Zuse's last wartime computer, the Z4, had the length of its words increased to 32 bits. It was evacuated to Göttingen as the Allies approached Berlin. Eventually it ended up in Basle, Switzerland, where it operated until 1954 — one of the most important computers working in Europe at the time.

Zuse was unable to manufacture computers in post-war Germany, so he concentrated on the theory of computers. He developed a sophisticated language called Plankalkül that could deal logically with both mathematics and more general information. When he was able to manufacture computers again he formed the Zuse Company, which was Germany's major computer manufacturer until 1969, when it was absorbed into the Siemens Corporation. Professor Zuse is still working in the computer industry.

Doodle Bug
Zuse's computers were developed to replace teams of technicians working with slide rules on aeronautical calculations. In particular, they were applied to the design of the V1 and V2 (pictured) flying bombs used so heavily in the Second World War

© THE IMPERIAL WAR MUSEUM

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

THE HOME COMPUTER COURSE BINDER

Now that your collection of Home Computer Course is growing, it makes sound sense to take advantage of this opportunity to order the two specially designed Home Computer Course binders.

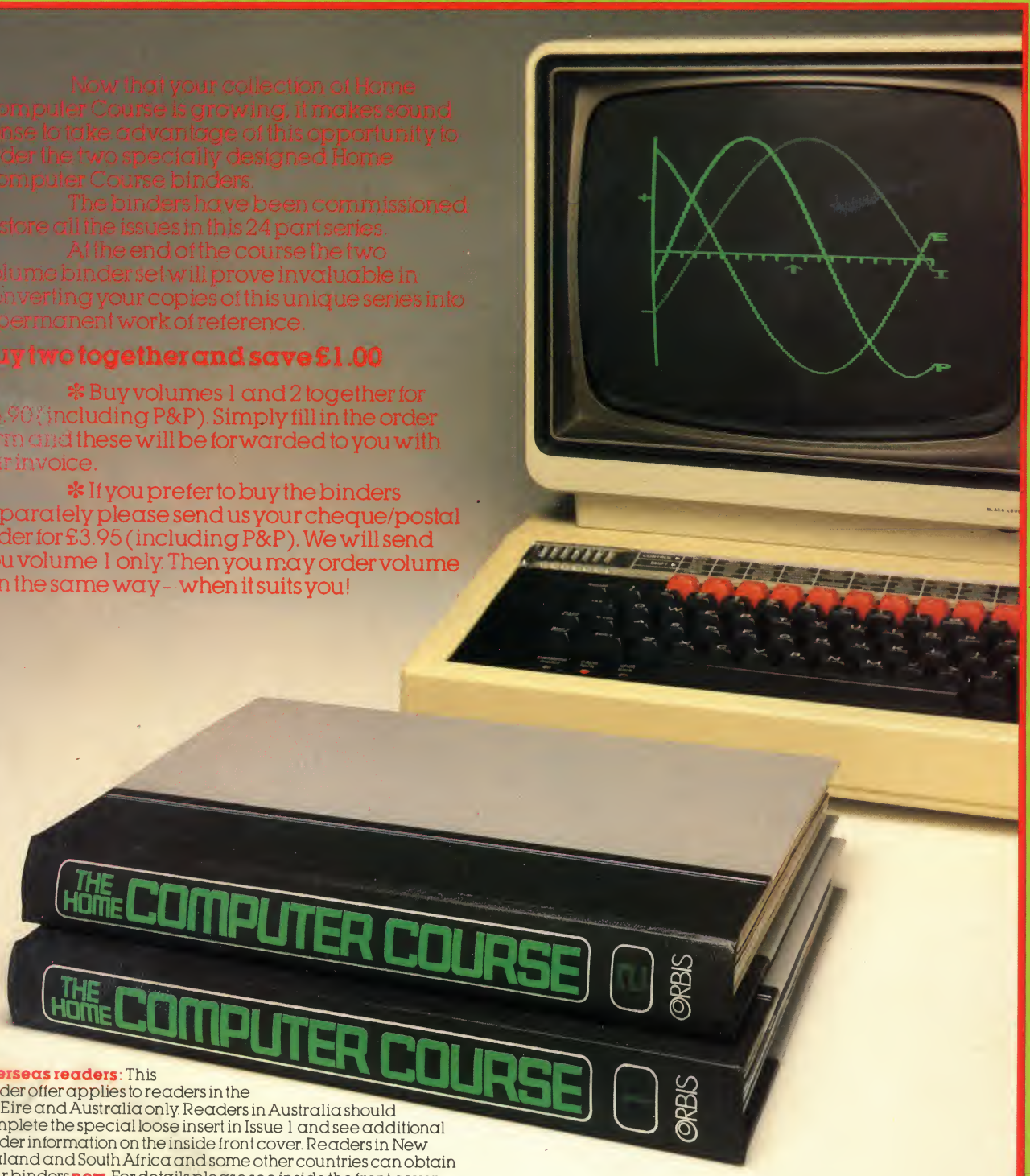
The binders have been commissioned to store all the issues in this 24 part series.

At the end of the course the two volume binder set will prove invaluable in converting your copies of this unique series into a permanent work of reference.

Buy two together and save £1.00

* Buy volumes 1 and 2 together for £6.90 (including P&P). Simply fill in the order form and these will be forwarded to you with our invoice.

* If you prefer to buy the binders separately please send us your cheque/postal order for £3.95 (including P&P). We will send you volume 1 only. Then you may order volume 2 in the same way - when it suits you!



Overseas readers: This binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in Issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain their binders **now**. For details please see inside the front cover.

Binders may be subject to import duty and/or local tax.

NEXT TO YOUR COMPUTER... YOUR COURSE MANUALS